IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application

Applicant(s):	Lorraine Denby et al.
Case:	503021-A-01-US (Denby)
Serial No.:	10/643,275
Filing Date:	August 19, 2003
Group:	2616
Examiner:	Kevin D. Mew
Title:	Method and Apparatus for Automatic Determination of Performance Problem Locations in a Network

DECLARATION OF PRIOR INVENTION UNDER 37 C.F.R. §1.131

We, the undersigned, hereby declare and state as follows:

1. We are named joint inventors of the invention that is the subject of the above-referenced U.S. patent application. We have assigned our respective interests in the patent application to Avaya Inc. ("Avaya") or a subsidiary thereof.

2. The invention was conceived and reduced to practice at some time prior to September 16. 2002. On or about September 16, 2002, a description of the invention and its actual reduction to practice was prepared in the form of an Avaya proprietary document entitled "Inferring Network Link Level QoS Indicators from End-to-End Measurements." Copies of the document and its associated cover sheet dated September 16, 2002 are attached hereto as Exhibit 1. The document is identified on its associated cover sheet as Avaya internal technical report number ALR-2002-037.

3. The Avaya proprietary document in Exhibit 1 describes an invention falling within one or more of the claims of the present application. For example, the description at pages 3-4, section 3, describes an approach in which test communications are generated in accordance with a selected pattern, end-to-end path measurement data is collected, and the end-to-end path measurement data is transformed to produce performance indicators for non-end-to-end paths. We reduced this invention to practice by implementing it in an actual network, referred to in the Avaya proprietary document as the BHD network. The reduction to practice is described in the Avaya proprietary document at pages 11-17, section 4.

4. On or about September 20, 2002, the undersigned inventor Bengi Karacali sent the Avaya proprietary document and its associated cover sheet as shown in Exhibit 1 to Martina Sharp, a secretary at Avaya, for further processing consistent with Avaya business practices regarding such documents. A copy of the email from Ms. Karacali to Ms. Sharp dated September 20, 2002 is attached hereto as Exhibit 2. The attachments to this email were the Avaya proprietary document and its associated cover sheet as shown in Exhibit 1.

5. On or about April 23, 2003, a summary of the invention was provided via email by Ms. Karacali to Brian K. Dinicola, an in-house patent attorney at Avaya. Mr. Dinicola subsequently determined that an invention submission should be opened, and directed that the submission be sent to Joseph B. Ryan, an outside counsel patent attorney, for preparation and filing of a patent application based on the submission. Mr. Ryan proceeded to prepare the above-noted patent application, and filed it on August 19, 2003. Copies of the invention summary and related emails are attached hereto as Exhibit 3.

6. All statements made herein of our own knowledge are true, and all statements made on information and belief are believed to be true.

7. We understand that willful false statements and the like are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001, and may jeopardize the validity of the application or any patent issuing thereon.

Date: 8/16/07

Lorraine Denby

Date: 08/16/2007

Bengi Karacali

Date: 8/16/2007

Jean Meloche

EXHIBIT 1

AVAYA LABS RESEARCH TECHNICAL REPORT

Date: 09.16.2002

Number: ALR-2002-037
Pages: 18

Title: Inferring Network Link Level QoS Indicators from End-to-End Measurements

Authors	Email Address	Phone	Company/Location
Lorraine Denby	ld@avaya.com	908 696 5112	Avaya, Basking Ridge, NJ
Bengi Karacali	karacali@avaya.com	908 696 5146	Avaya, Basking Ridge, NJ
Jean Meloche	jmeloche@avaya.com	908 696 5144	Avaya, Basking Ridge, NJ

Keywords: IP Telephony, Quality of Service, Network Assessment

Abstract

Multimedia applications such as IP Telephony are among the applications, which demand strict quality of service (QoS) guarantees from the underlying data network. Understanding the network behavior at all stages of the deployment of such applications is critical for their performance. In this paper we describe an approach to understanding link level QoS behavior in a network. Even though this information is potentially useful for many purposes, in this paper we focus on using the link level information for the purpose of locating the sources of performance problems in a network. We present an empirical study conducted on a real network spanning three geographically separated Avaya sites. The empirical results indicate that our approach efficiently and accurately pinpoints links in the network incurring the most significant delay.

Author(s): I/we request release for intended external publication in: □ Yes ☒ No

Lorraine Denby Bengi Karacali Jean Meloche

Approvals: I have reviewed and, if Yes is checked above, recommend release for external publication.

Chandra Kintala Jim Landwehr

Approved for External Release? □ Yes □ No
(If No, treat as Avaya Proprietary, pursuant to company instructions)

Ravi Sethi Date

Inferring Network Link Level QoS Indicators from End-to-End Measurements *

L. Denby, B. Karaçalı, J. Meloche
Avaya Labs Research
233 Mount Airy Road, Basking Ridge, NJ 07920

Abstract

Multimedia applications such as IP Telephony are among the applications, which demand strict quality of service (QoS) guarantees from the underlying data network. Understanding the network behavior at all stages of the deployment of such applications is critical for their performance. In this paper we describe an approach to understanding link level QoS behavior in a network. Even though this information is potentially useful for many purposes, in this paper we focus on using the link level information for the purpose of locating the sources of performance problems in a network. We present an empirical study conducted on a real network spanning three geographically separated Avaya sites. The empirical results indicate that our approach efficiently and accurately pinpoints links in the network incurring the most significant delay.

1 Introduction

Recent trends in the industry towards unified communications emphasize the need for converged networks to deliver acceptable quality of service(QoS) for different types of applications with varying QoS needs. Multimedia applications such as IP Telephony are among the end-to-end applications which demand strict QoS guarantees from the underlying data network. Understanding the network behavior at all stages of the deployment of such applications is critical for their performance. For example, at the pre-deployment stage, it is necessary to assess whether the network can deliver the required QoS and more importantly which parts of the network fail to do so. After deployment, monitoring the performance of the network is necessary for maintaining acceptable QoS levels. In this paper we describe an approach to understanding link level QoS behavior in a network. Even though this information is potentially useful for many purposes, in this paper we focus on using the link level information for the purpose of locating the sources of performance problems in a network.

The main objective of our approach is to assign performance estimates to links or groups of links in the network. In the context of locating the source of performance problems, one can imagine that a single bottleneck link may cause performance problems on any end-to-end path traversing the link. Similarly, a number of bottlenecks can show symptoms throughout the network. The amount of effort to locate the bottlenecks as well as their scale may be significant depending on the network topology. Our approach automates this task by providing a mechanism to understand the link level behavior in the network. Based on a user defined threshold of acceptable performance, those links that have unacceptable levels may be investigated further to understand the nature of the problems. The ability to accurately pinpoint the network regions with unacceptable performance levels constitutes a measure of accuracy for our approach.

Our approach requires that the network topology is available and software agents are installed throughout the network that can collect end-to-end performance measurements such as end-to-end delay, jitter, and packet loss. Both of these tasks are accomplished by our prototype ExpertNet^{TM}. ExpertNet^{TM}is a tool that collects various measurements from a network including topology information, periodic traffic measurements from network devices via SNMP, and end-to-end performance measurements with the use of software agents installed in the network. Our approach starts with analyzing the network topology with respect to the locations of software agents for collecting measurements. We search the network topology for an *optimum* set of network paths that can be assigned performance estimates. Note that, informally, the *optimum* set is the one composed of as short and as disjoint paths as possible. In cases where an *optimum* solution is costly, we select a set of paths that is a low cost approximation but potentially decreases the accuracy of our approach. Then, we determine where in the network to collect end-to-end measurements such that we can determine performance estimates for the paths in the selected set. Using ExpertNet^{TM}, we collect such measurements. Finally, we analyze the collected data in light of the topology and infer performance estimates on links or groups of links.

*The authors thank Mark Bearden, Scott Bianco, David Stott, and the ExpertNet^{TM}development team for their contributions to this work.

Our contributions are threefold: First we provide an algorithm that analyzes the network topology and provides a set of paths to which performance estimates can be assigned. This algorithm also provides the set of end-to-end paths on which measurements should be collected to infer link level performance estimates for paths in the selected set. Second, we formally prove that when the network topology is a tree, this algorithm determines the *optimum* set of paths. Note that our approach considers the physical topology hence it is expected that the topology is either a tree or close to being a tree. Third, we present an empirical study conducted on a real network spanning three geographically separated Avaya sites. The empirical results indicate that our approach efficiently and accurately pinpoints links in the network incurring the most significant delay. Furthermore, our approach is more accurate in comparison to a completely random approach of measurement collection.

In section 2 we provide the background information, notation, and formally define the problem. In section 3 we describe the steps of our approach. In section 4 we illustrate the application of our approach on a real network, provide promising preliminary results, and discuss the findings of the results. Finally, in section 5 we present the conclusions.

2 Preliminaries

In this section we describe the notation and the definitions used throughout the paper.

Definition 2.1 *Let $G = (D, L)$ be a a connected network topology graph where D is a set of nodes and L is a set of links. Each device in D is of type router or switch. Let D_1 and D_2 be two distinct devices in D. There is a link between D_1 and D_2 if and only if there is a direct communication path between D_1 and D_2.*

Path in G is a sequence of links from L. Length of a path p, denoted as $length(p)$ is the number of links on p. A link is a path of length 1. ϵ denotes a path of length 0. For two paths x and y in G, $x.y$ denotes the path formed by the concatenation of x and y in G. A path x of G is considered a subpath of another path y in G iff there exists a path z such that $x.z$=y. If $z \neq \epsilon$, then x is a proper subpath of y.

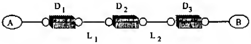


Figure 1. Call Path

Figure 1 illustrates the network elements involved in a sample voice call between endpoints A and B. We refer to a device that can initiate and respond to voice calls as an endpoint.

For our purposes, in addition to the ability to generate RTP streams, an endpoint also measures the QoS statistics pertaining to these streams. We refer to the end-to-end traffic between A and B as a flow.

We assume the path from A to B is the reverse of the path from B to A. A line between two devices denotes a bidirectional edge. Assume that the voice packets pass through routers indicated as D_1, D_2, and D_3 as in Figure 1. The router interfaces that the packets traverse are also marked on the figure with circles. For the call shown in Figure 1, the call path is $L_1.L_2$.

We assume that the network topology is static and the paths between endpoints do not change through time. Note that this is a reasonable assumption for enterprise networks. Furthermore, we assume that the path between two endpoints is the same in both directions.

Definition 2.2 *A device in G that has an endpoint attached is a leaf. Set $E \subset D$ denotes the set of leaves in G.*

In Figure 1, D_1 and D_3 are the leaves.

Definition 2.3 *A path in G that lies between leaves is considered as an end-to-end path.*

In Figure 1, path $L_1.L_2$ is an end-to-end path.

Definition 2.4 *Set P for a given G and E denotes the set of all end-to-end paths in G between devices in E.*

Set P is computed from G and E. The part of the topology that we can collect measurements from lies between the endpoints, i.e. P. In the rest of this paper we focus only on the paths between the endpoints.

Definition 2.5 *A matrix where rows denote end-to-end paths and columns denote sequence of links (or paths) in G is considered a flow matrix. Let M be an $n \times m$ flow matrix. For $0 < i \leq n$ and $0 < j \leq m$:*
$m_{i,j}$ = number of times the end-to-end path in row i traverses the path in column j
A flow matrix formed by P and L is considered the complete flow matrix for G and E.

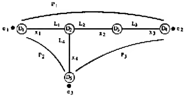


Figure 2. Example 1

Consider Example Network 1 shown in Figure 2. In this sample network there are 5 network devices, D=$\{D_1, D_2, D_3, D_4, D_5\}$ and 4 links L=$\{L_1, L_2, L_3, L_4\}$. Leaves D_1, D_4, and D_5 are marked by a black dot next to the device in the figure. P=$\{P_1, P_2, P_3\}$ where $P_1 = L_1.L_2.L_3$, $P_2 = L_1.L_4$, and $P_3 = L_4.L_2.L_3$.

Two example flow matrices for Example Network 1 are shown in Figure 3. Case 1 corresponds to end-to-end paths P_1 and P_2. Case 2 corresponds to end-to-end paths P_1, P_2, and P_3.

$$\left[\begin{array}{c|cccc} & L_1 & L_2 & L_3 & L_4 \\ \hline P_1 & 1 & 1 & 1 & 0 \\ P_2 & 1 & 0 & 0 & 1 \end{array}\right]$$

Flow matrix 1

$$\left[\begin{array}{c|cccc} & L_1 & L_2 & L_3 & L_4 \\ \hline P_1 & 1 & 1 & 1 & 0 \\ P_2 & 1 & 0 & 0 & 1 \\ P_3 & 0 & 1 & 1 & 1 \end{array}\right]$$

Flow matrix2

Figure 3. Flow Matrices

Definition 2.6 *A call pattern is a list of endpoint pairs, that concurrently collect measurements from a given network pertaining to end-to-end paths between the endpoint pairs in the list.*

The call pattern may have other information pertaining to the measurements such as duration etc. These parameters are not important in this discussion so for simplicity we assume that exercising a call pattern corresponds to collecting simultaneous performance metrics from the network for a user defined duration.

For each flow matrix, a corresponding call pattern can be computed. If there is one endpoint per leaf, then the call pattern for a given flow matrix is unique. If there are more than one endpoints attached to a leaf, then multiple call patterns may match the same flow matrix. In Example Network 1, call pattern $\{(e_1, e_2), (e_1, e_3)\}$ corresponds to flow matrix 1. Similarly, call pattern $\{(e_1, e_2), (e_1, e_3), (e_2, e_3)\}$ corresponds to flow matrix 2.

Exercising a call pattern corresponds to collecting simultaneous measurements from the network. This is necessary since we assume that the measurements reflect the same network states.

3 Approach

Our approach involves collecting end-to-end measurements from the network under consideration at regular time intervals and then using the measurements to infer link level performance estimates at each time interval. Measurement collection requires injecting synthetic traffic between endpoints in the network. Note that the time interval length is an adjustable parameter that depends on how frequently one would like to sample the network.

Formally, at each time interval our measurements form the following linear equation:

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \varepsilon \tag{1}$$

where :

- **y** is a vector of measurements collected on end-to-end paths such as delay
- **A** is a flow matrix
- **x** is a vector of network link level performance metrics
- ϵ is random error

In order to determine the **x** values at each time interval, we solve the above linear equation. The size of A is an issue in our approach. Note that, the flow matrix A determines which paths we can compute performance metrics. Hence there is a need to ensure that the columns of flow matrix A correspond to as short length and as disjoint paths as possible. Furthermore, A is typically singular which constitutes a challenge.

In order to address the issue of flow matrix size, we take advantage of the simple observation that links carrying the same flow can be treated as one. In Example 1, by recognizing that links L_2 and L_3 carry the exact same flows and hence can be treated as one unbreakable path, the flow matrices could be reduced by one column. Section 3.1 describes the application of this observation to reducing the network topology.

As mentioned above, the call pattern and the corresponding flow matrix exercised impacts the granularity of our link level estimates. In Example Network 1, let the average delay incurred on each link at a given time interval be $x_1, x_2, x_3; x_4$ for L_1, L_2, L_3, L_4 respectively. Using flow matrix 1 $x_1 + x_2 + x_3$ and $x_1 + x_4$, namely, average delay attributable to paths $L_1.L_2.L_3$ and $L_1.L_4$ can be estimated. Using flow matrix 2, x_1, $x_2 + x_3$ and x_4, namely, average delay attributable to paths L_1, $L_2.L_3$, and L_4 can be estimated.

Equations with Flow matrix 1

$$x_1 + x_2 + x_3 = y_1 \qquad x_1 + x_4 = y_2 \qquad \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Equations with Flow matrix 2

$$x_1 + x_2 + x_3 = y_1 \qquad x_1 + x_4 = y_2 \qquad x_2 + x_3 + x_4 = y_3 \qquad \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

As the example illustrates, the columns of the flow matrix are the paths we can provide estimates for. In order to estimate performance metrics for each link in the network, it is sufficient to simply exercise a call pattern pertaining to the complete flow matrix. However this approach would not be scalable since it requires a large call pattern . The two major impacts of such a large call pattern is that each endpoint participates in many concurrent synthetic calls increasing the load on the endpoint and the total traffic injected to the network is significant.

As the example above illustrates, the selection of flow matrix and a corresponding call pattern is the key in our approach. We address the issue of selecting a non-singular flow matrix that provides granular link level performance estimates in 3.2.

We discuss the measurement collection step in detail in section 3.3.

3.1 Network Topology Reduction

Definition 3.1 *For a given G and P, a sequence of links is defined as a pipe if each link in the sequence carries the same flow. Set I for a given G and E denotes the set of pipes for a given G and E.*

Note that a pipe is a path. We refer to the number of links on a pipe as the length of the pipe .

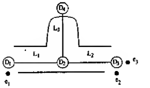
Algorithm Generate_Pipes shown in Figure 4 computes the pipe set, I, for a given topology, G and an endpoint set, E. The algorithm starts with constructing P, the set of all end-to-end paths between devices in E. Let M be the complete flow matrix for G with respect to E. The algorithm then groups links in L into disjoint sets such that all links in a given set have the same column vector in M. Let $\mathbf{S} = \{S_1, S_2, \ldots S_k\}$ be the result of this grouping assuming there are k distinct column vectors in M. (Sets of sets are denoted in boldface.) Note that each element S_i of **S** is the set of links that carry the same flow. Finally, the algorithm constructs a path for each S_i that corresponds to the concatenation of links in S_i. The path for each S_i is added to I. In cases where the topology contains loops, links grouped into a set may not be consecutive to form a path. In such cases, each link is considered a pipe by itself and added individually to I.

Figure 5 illustrates the case when there is a loop in the graph. Assume D_4 is a router and the rest of the devices in the topology are switches. Assume the path between endpoints e_1 and e_2 is $L_1.L_2$ and the path between e_1 and e_3 is $L_1.L_3.L_3.L_2$. Assume endpoints e_1 and e_2 are on the same vlan and endpoint e_3 is on a different vlan than e_1 and e_2.

```
Generate_Pipes($G = (D, L)$:Network Topology Graph, $E$: Set of Leaves)
$I$: Set of pipes in $G$ wrt $E$
$I \leftarrow \emptyset$
Compute $P$ for $G$ wrt $E$
Let $M$ be the complete flow matrix for $G$ and $P$
// Group links with the same column vector into disjoint sets
Let $k$ be the number of distinct column vectors in $M$
Form a set $\mathbf{S} = \{S_0, S_1, ..., S_k\}$ where :
    each $S_i, 0 < i \leq k$ contains links in $L$ with the $i^{th}$ distinct column vector in $M$
// Ensure that links in each element of S form a path in G
for $i$=1 to |S|
    if links in $S_i$ are consecutive and form a path
        then merge $S_i$ into path p, $I \leftarrow I \cup \{p\}$        // add the path formed by the links in $S_i$ as a pipe
        else $I \leftarrow I \cup S_i$                                 // add each link as a pipe by itself
return $I$
```

Figure 4. Algorithm Generate_Pipes

Assume D_4 is a router that switches between Vlans. The complete flow matrix for this example is also shown in the figure. Algorithm Generate_Pipes computes $I = \{L_1, L_2, L_3\}$. In this case L_1 and L_2 carry the same flows even though they are not on the same pipe. The last part of algorithm Generate_Pipes ensures that links L_1 and L_2 are not placed in the same pipe even though they have the same column vectors in M as shown in Figure 5. In this example the reduced topology G' is the same as G since the pipe set I is the same as the link set L.



$$\left[\begin{array}{c|ccc} & L_1 & L_2 & L_3 \\ \hline e_1 - e_2 & 1 & 1 & 0 \\ e_1 - e_3 & 1 & 1 & 2 \\ e_2 - e_3 & 0 & 0 & 0 \end{array}\right]$$

Figure 5. Loop Example

After the computation of pipe set I by Algorithm Generate_Pipes, the reduced network topology is denoted as $G' = (D', I)$ where $D' \subset D$ and there is a pipe in I between each device in D'. Note that E is the same for G and G' since all leaf nodes are included in the reduced topology graph. Furthermore, the topology graph is connected after the reduction. Set P' for a given G' and E denotes the set of all end-to-end paths in G' between devices in E.

Example Network 1 of Figure 2 after the reduction becomes $G' = (D', I)$ where $D' = \{D_1, D_2, D_4, D_5\}$ and $I = \{L_1, L_2.L_3, L_4\}$.

Lemma 3.1 *If G is a tree, then in the reduced topology graph $G' = (D', I)$, each device is either a leaf or has degree of at least 3.*

Proof:

Assume there is a device $D_i \in D'$ such that D_i is not a leaf and has degree < 3. Since the reduced graph is connected every node has degree at least 1.

Case 1: D_i has degree 1.

In this case D_i is the last node on an end-to-end path and hence must be a leaf which is a contradiction.

Case2: D_i has degree 2.

Since D_i has degree 2, there must be exactly two neighbor devices to D_i. Let D_x and D_y be the distinct neighbors of D_i. Note that if $D_x = D_y$ then D_i is on a loop. Since devices and links that form loops are replaced with pipes in the reduced topology, D_x and D_y must be different.

Figure 6 illustrates this case. Note that c_x and c_y must be distinct pipes in the reduced topology. Since D_i is not a leaf, there cannot be a flow that starts or ends in D_i. This means flows on c_x and c_y must be equal. In this case c_x and c_y must be on the same pipe which is a contradiction.

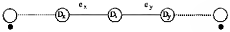


Figure 6. Illustration for Lemma

3.2 Call Pattern Selection

Definition 3.2 *Let $c = L_i. \;\; \ldots \;\; .L_k$ be a path in $G = (D, L)$. $v(c)$ denotes an additive QoS measurement on c and $v(c) = v(L_i) + \ldots + v(L_k)$.*

Definition 3.3 *A network path $c = L_i. \;\; \ldots \;\; .L_k$ of $G = (D, L)$ and P is considered estimable if there is a non-singular flow matrix A of end-to-end paths from P and paths in G such that c corresponds to a column in A.*

The above definition implies that there exists a flow matrix for any estimable parameter. Let A be the flow matrix for c. Trivially following from the above definition, if $v(c)$ is a parameter in x and $\mathbf{y} = \mathbf{A}\mathbf{x} + \epsilon$ is solvable for $v(c)$ then c is estimable.

Definition 3.4 *A set of paths is considered minimal if no path in the set contains another member of the set as a proper subpath.*

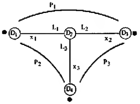


Figure 7. Example 2

Consider Example Network 2 shown in Figure 7. In this example there are 4 network devices, D=$\{D_1, D_2, D_3, D_4\}$, and 3 links, L=$\{L_1, L_2, L_3\}$. Devices D_1, D_3, and D_4 are connected to an endpoint as indicated by an black dot next to the device in the figure. P=$\{P_1, P_2, P_3\}$ where $P_1 = L_1.L_2$, $P_2 = L_1.L_3$, and $P_3 = L_2.L_3$.

Consider flow matrix 1 with paths P_1 and P_2 and flow matrix 2 with paths P_1, P_2, and P_3.

Flow matrix 1

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Using flow matrix 1, the minimal set of estimable paths is $\{L_1.L_2, L_1.L_3\}$. Similarly, using flow matrix 2, the minimal set of estimable paths is $\{L_1, L_2, L_3\}$. Given that our goal is to estimate QoS parameters such as delay as granular as possible, clearly flow matrix 2 is more advantageous.

Definition 3.5 *Set of worms , W, for a given G and P, is the minimal set of paths where each member path is estimable and the union of links on all member paths is L.*

Note that the set of worms for a given G and E is a minimal set. Furthermore, there does not exist a path p in any set of estimable paths for a given G and E, such that p that is a proper subpath of a path in W. For the topology and the leaf set of Example 2, $W = \{L_1, L_2, L_3\}$.

Definition 3.6 *A flow matrix M for G' and E is optimum if*

1. *the columns of M are worms*
2. *there does not exist a flow matrix M' such that columns of M' are worms and number of rows of M' is less than the number of rows of M.*

Note that optimum flow matrix is non-singular. Our flow matrix selection strategy searches for optimum flow matrices.

Ideally, the pipe set I for G and E is the worm set for G and E, i.e. $I = W$. This is not always possible. Consider Example 3 in Figure 8. In this case, the current location of endpoints allow for three end-to-end paths, P_1, P_2, and P_3. For this example, $W = \{L_1.L_2.L_3, L_1.L_4.L_6, L_3.L_5.L_6\}$ same as the end-to-end paths.

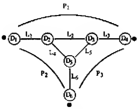


Figure 8. Example 3

In order to select an optimum flow matrix for a G and E, at least $|I|$ end-to-end paths are necessary. This follows from requiring n equations to solve for n unknowns. Note that even in cases where $I = W$, not any $|I|$ end-to-end paths are sufficient to achieve the optimum flow matrix.

Consider Example 4 shown in Figure 9 with $|I| = |W| = 4$. $I = W = \{L_1, L_2, L_3, L_4\}$. In this case there are at least two flow matrices that yield two distinct sets of estimable paths. Figure 9 shows two flow matrices and the minimal set of paths that can be estimated using each flow matrix.

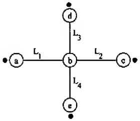


Figure 9. Example 4

$$\left[\begin{array}{c|cccc} & L_1 & L_2 & L_3 & L_4 \\ \hline L_1.L_4 & 1 & 0 & 0 & 1 \\ L_4.L_2 & 0 & 1 & 0 & 1 \\ L_2.L_3 & 0 & 1 & 1 & 0 \\ L_3.L_1 & 1 & 0 & 1 & 0 \end{array}\right]$$

$$\{L_1.L_4, L_4.L_2, L_2.L_3, L_3.L_1\}$$

$$\left[\begin{array}{c|cccc} & L_1 & L_2 & L_3 & L_4 \\ \hline L_1.L_2 & 1 & 1 & 0 & 0 \\ L_1.L_3 & 1 & 0 & 1 & 0 \\ L_1.L_4 & 1 & 0 & 0 & 1 \\ L_2.L_3 & 0 & 1 & 1 & 0 \end{array}\right]$$

$$\{L_1, L_2, L_3, L_4\}$$

An intuitive algorithm to flow matrix selection would be building a non-singular flow matrix gradually. Let M be a flow matrix constructed by adding rows corresponding to paths in P'. A path p in P' gets selected if adding a new row corresponding to p increases the rank of M. The algorithm stops when all paths are processed. One obvious obstacle in this intuitive approach is what constitutes the columns in M. An obvious choice is the set of pipes. However in this case the algorithm still has to determine the worms.

Another issue is that the algorithm may select unnecessary paths since the solution is dependent on the order in which the paths in P' are selected. Many path sets of different sizes may result with the same flow matrix. Similarly, paths sets of the same size may result with different flow matrices as shown in Figure 9. In the example of Figure 9, flow 1 requires additional paths to produce the same estimable path set as flow matrix 2.

The final issue pertains to satisfying various constraints in flow matrix selection. This algorithm cannot handle cases where there are other constraints on matrix selection such as ensuring that certain pipes in the network are estimable. There is a need to be able to answer the following question: "is path s" estimable for a given G' and P'.

Algorithm Select_Matrix shown in Figure 10 overcomes the issues described above. The algorithm is a heuristic for generating a flow matrix for a given G' and E. Set of worms , W, denote the columns and set of end-to-end paths, R, denote the rows in the flow . However, for network topologies where there exists a unique path between each node, the algorithm is guaranteed to produce a flow matrix whose columns are the worm set for G' and E.

The algorithm maintains a list of paths, *open*, which contains paths with no estimable parts. As pipes become estimable during the course of the computation, each path p in *open* is modified by removing estimable pipes from p. The original end-to-end path for each path in *open* is maintained.

```
Select_Matrix($G' = (D', I)$:Reduced Network Topology Graph, $E$: Set of Leaves)

$W$: Set of worms in $G'$ wrt $E$, $W \leftarrow \emptyset$
$R$: Set of paths, $R \leftarrow \emptyset$
Compute $P'$ for $G'$ wrt $E$
$open \leftarrow P'$
while $open \neq \emptyset$
    select $p$ from $open$
    for each pipe $c_i$ on $p = c_1.c_2.\dots.c_{length(p)}$
        $\boxed{\text{if } \exists S \subset open \text{ such that } S \text{ makes } c_i \text{ estimable}}$          $\leftarrow$ B1
            Compute $S'$ which has the original value of each path in $S$
            $R \leftarrow R \cup S'$
            $W \leftarrow W \cup \{c_i\}$
            update $open$ and $W$ such that $\forall p' \in open$
                $p'$ does not contain any estimable path in $W$          // $c_i$ is removed from paths in $open$
        else
            $c_{i+1} \leftarrow c_i.c_{i+1}$
    $open \leftarrow open \setminus \{p\}$
return $W, R$
```

Figure 10. Algorithm Select_Matrix

At the beginning, P' for input G' and E is computed and *open* is initialized to P'. At each step, until paths in *open* are exhausted, the algorithm first removes a path p from *open*. Then for each pipe c_i on p, it searches *open* for a set of paths S that can estimate c_i. If such a set S is not found, that means c_i is not estimable so c_i is appended to the next pipe on the path. The next pipe processed becomes $c_i.c_{i+1}$. If however, c_i is estimable by set S, then c_i is added to W. Let S' be the set of original paths pertaining to paths in S. S' is added to R. Then *open* is updated to reflect the fact that c_i is now estimable. This involves, removing c_i from all paths in *open*. After this operation, *open* is sorted so that partial paths, i.e. those that have been reduced earlier are moved to the top and will have preference for processing. When all pipes on p are processed, p is removed from *open*.

Note that the operation of eliminating estimable pipes from paths in *open* preserves Lemma 3.1 for tree topologies. Assume c is a pipe that just became estimable and assume c is between devices D_x and D_y on path p. Assume D_{xy} is the new device formed by the removal of c and the merging of D_x and D_y. If either D_x or D_y were a leaf before removal of c then D_{xy} is also a leaf after the operation. If neither D_x or D_y were a leaf before the removal of c then after the operation the new node, D_{xy} will have degree $degree(D_x) + degree(D_y) - 2$ which is at least 4 .

The box labeled as B1 in Figure 10 marks the step where the computation of estimating set S for c_i occurs. Figure 11 illustrates three basic patterns for estimating pipe c_i. Dashed lines in the figure indicate paths that are in the topology but not immediately relevant to the argument. This step searches the paths in open to determine whether any of the three patterns above apply to paths in *open*.

In pattern 1, there is a path P_1 in *open* that equals c_i. In this case $v(c_i) = v(P_1)$ and $S = \{P_1\}$. c_i is estimable trivially. In pattern 2, P_1 and P_2 are in *open* such that $P_2.c_i = P_1$. $v(c_i) = v(P_1) - v(P_2)$ and $S = \{P_1, P_2\}$. The flow matrix where S are the rows and $\{P_2, c_i\}$ are the columns estimate c_i. In pattern 3, P_1, P_2, and P_3 are in *open* where $P_1 = c_i.x$, $P_2 = c_i.y$, and $P_3 = x.y$. $v(P_1) = v(c_i) + v(x)$, $v(P_2) = v(c_i) + v(y)$, and $v(P_3) = v(x) + v(y)$. In this case $v(c_i) = \frac{v(P_1)+v(P_2)-v(P3)}{2}$. Hence the flow matrix where S are the rows and $\{c_i, x, y\}$ are the columns estimate c_i.

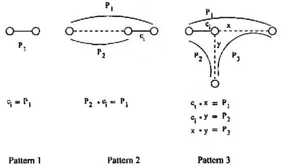


Figure 11. Basic Patterns

Lemma 3.2 *Let $G' = (D', I)$ be a connected reduced network topology where there is exactly one path between any two devices in D'. Algorithm Select_Matrix will select the pipe set I as the worm set for G' and E.*

Proof: Assume there is a pipe in I which is not in W produced by Algorithm Select_Matrix. Let p be the path that the algorithm processed but could not find S for estimating a pipe. Assume c is the first pipe on p that could not be estimated. Let $p = x.c.y$.

Case1: $x = \epsilon$ and $y = \epsilon$. Then $p = c$.

In this case $length(p) = 1$ and c is estimable by pattern 1. This is a contradiction.

Case 2: $x = \epsilon$. Then $p = c.y$.

Assume D_x and D_y are the ends of c as shown in Figure 12 case 2. By Lemma 3.1, D_y is either a leaf or the degree of D_y is at least 3.

If D_y is a leaf then c is estimable with paths $\{c.y, y\}$ and worms $\{c, y\}$ by pattern 2. Hence a contradiction.

If D_y is not a leaf but the degree of D_y is at least 3 then paths $c.y$ and $z.y$ must also be in *open*. Then $v(c) = \frac{v(c.y)+v(c.z)-v(z.y)}{2}$. Hence c is estimable with paths $\{c.y, c.z, z.y\}$ and worms $\{c, x, y\}$ by pattern 3. Hence a contradiction.

Case 3: $y = \epsilon$. $p = x.c$.

Assume D_x and D_y are the ends of c as shown in Figure 12 case 3. By lemma 3.1, D_x is either a leaf or D_x must have degree at least 3.

Algorithm Select_Matrix will process x in $length(x)$ steps. Since c is the first pipe on p that could not be estimated, all pipes on path x are estimable. At the end of $length(x)$ steps, p becomes c. The contradiction follows from Case 1 above.

Case 4: $p = x.c.y$.

Assume D_x and D_y are the ends of c as shown in Figure 12 case 4. By lemma 3.1, D_x is either a leaf or have degree at least 3. Same holds from D_y.

Algorithm Select_Matrix will process x in $length(x)$ steps. Since c is the first pipe on p that could not be estimated, all pipes on path x are estimable. At the end of $length(x)$ steps, p becomes $c.y$. The contradiction follows from Case 2 above.

In cases where the network topology has cycles, then algorithm Select_Matrix cannot guarantee that the worm set for G' and E will be the columns of the selected flow matrix. In these cases, the algorithm provides an approximation. In such cases determination of the optimum solution requires enumerating every possible path in the topology, checking to see if it is estimable, and then taking the minimal set of estimable paths. Clearly this is a costly approach with marginal benefits.

Selection of a call pattern is trivial if there is only one endpoint per leaf and there are no additional constraints. In this case, each end-to-end path corresponding to a row of the flow matrix selected by algorithm Select_Matrix corresponds to a unique endpoint pair. The pairs whose paths correspond to the flow are selected as the call pattern to be used for data collection.

In cases when there are multiple endpoints per leaf, selecting pairs by distributing calls evenly reduces the load on each endpoint. For instance every time a leaf appears on an end-to-end path, a different endpoint attached to the leaf may be used to inject the synthetic traffic to distribute the load.

In order to satisfy various constraints on call pattern selection, algorithm Select_Matrix needs to take into consideration that paths are selectable only if they satisfy the required conditions. This requires modifying box B1 in the algorithm Select_Matrix by adding additional constraints. Assume it is required that each endpoint participates in a maximum of N calls at the same time. This implies that an endpoint can appear maximum of N times in the selected call pattern . In this

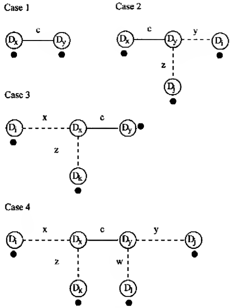


Figure 12. Illustration for Lemma

case, additional data structures associated with a path can maintain information about the call pattern . After the selection of each path (i.e. participating in S) these structures are updated to reflect the new addition to the call pattern . Other constraints may be accommodated by modifying the function implemented in box B1. Note that when constraints are imposed on call pattern selection, optimum flow matrix may not be selected.

3.3 Network Measurement Collection & Analysis

Network measurements are collected by simultaneously injecting synthetic traffic between the endpoints in the selected call pattern . As mentioned earlier, data collection occurs at regular, user defined time intervals for a user defined duration. At each time interval t_i, synthetic traffic is injected between all endpoint pairs in the call pattern simultaneously. We rely on the ExpertNet tool described in [1] to exercise a given call pattern and to collect the end-to-end measurements.

Ideally, the same call pattern and hence the same flow matrix is exercised at each time t_i. Let y_i be the vector of end-to-end measurements collected at time t_i and x_i be the vector of network segment parameters. $\mathbf{y_i} = \mathbf{Ax_i} + \epsilon$. Since A is the same at each time interval, to obtain x values at each iteration, only different y values need to be substituted.

Empirical studies have shown that not all attempted synthetic calls of the selected call pattern succeed. There can be a number of reasons for a synthetic call to fail, including unfavorable network conditions, problems with endpoint software etc. In such cases, the equation becomes $\mathbf{y_i} = \mathbf{A_i x_i} + \epsilon$ where the rows of A_i is a subset of the rows of the selected flow matrix.

In order to address failed calls one strategy is to assume all failures are due to network congestion and assume infinite delay on all end-to-end paths where a call failed. By substituting infinite delay on y values pertaining to failed calls, we can use the same A matrix at each iteration and the processing becomes simple.

Even though this is a simple solution, it may be misleading since it is unlikely that the failure of calls is always due to network problems. Call failures due to reasons other than network problems are frequently encountered while using the third party software, which contributed to data collection. Hence, using the above approach would yield misleading results in the least.

A better solution would be to recompute a different A_i at each time t_i using the successful calls. Hence at each time interval t_i , it is necessary to compute a different flow matrix matching the successful calls of call pattern at t_i.

Algorithm Compute_EstPaths shown in Figure 13 computes M, the minimal set of estimable paths from a given set of end-to-end paths, P'_{t_i}, in the given topology. Note that flow matrix for successful calls at time t_i has rows P'_{t_i} and columns as computed by algorithm Compute_EstPaths.

Algorithm Compute_EstPaths follows the main outline as algorithm Select_Matrix in processing paths in *open* and ignores parts of the algorithm Select_Matrix that are relevant to end-to-end path selection; i.e. manipulation of R. Algorithm Compute_EstPaths processes paths until *open* becomes empty. Processing of a pipe that is found to be estimable is exactly the same as algorithm Select_Matrix. Upon encountering a non-estimable pipe c on path p, the algorithm, does not give up immediately but simply stops processing p and leaves it in *open* and continues processing the next path hoping that when it comes back to p, c is estimable. The algorithm converges if it is unsuccessful in finding any estimable pipe on all paths in *open*. In that case if there are any paths remaining in *open*, the shortest such path is removed from *open* and added to M. This is required to address cases where the sets remaining in *open* do not constitute a minimal set. Then algorithm searches for a solution using the updated estimable set.

The difference between the algorithms in manipulating *open* is due to an assumption change. Algorithm Select_Matrix assumes the following: Let c_i and c_{i+1} be two consecutive pipes on a path. If c_i is not estimable by itself, then the estimable path that contains c_i also contains c_{i+1}. This means that c_{i+1} cannot be estimable in isolation from c_i. This assumption does not hold when only a restricted subset of P' is used as is the case for algorithm Compute_EstPaths. It is possible that at the time c_i is processed, even though no paths that make c_i is found, paths that make c_{i+1} estimable can be found. Hence the algorithm does not immediately declare c_i not-estimable but continues searching for estimable pipes on other paths in *open*.

```
Compute_EstPaths($G' = (D', I)$):Reduced Network Topology Graph, $E$: Set of Leaves, $P'_{t_i}$: End-to-end paths at time $t_i$ )

$M$: A Minimal set of estimable paths for $G'$ wrt $E$, $M \leftarrow \emptyset$
$open \leftarrow P'_{t_i}$
while $open \neq \emptyset$
    while $open$ not converged
        select $p$ from $open$
        for each pipe $c_i$ on $p = c_1.c_2.\ldots.c_{length(p)}$
            $\boxed{\text{if } \exists S \subset open \text{ such that } S \text{ makes } c_i \text{ estimable}}$
                $M \leftarrow M \cup \{c_i\}$
                update $open$ and $M$ such that $\forall p' \in open$
                    $p'$ does not contain any estimable path in $M$ and          // $c_i$ is removed from paths in open
                    $open \leftarrow open \setminus \{p\}$
        else
            abort processing of $p$
    if $open \neq \emptyset$
        select shortest $p$ in $open$
        $open \leftarrow open \setminus \{p\}$
        $M \leftarrow M \cup \{p\}$
return $M$
```

Figure 13. Algorithm Compute_EstPaths

4 Empirical Results

The main goal of our experimental study is to first assess the ability of our approach in localizing the source of problems in a network as finely as possible. The second goal is compare the localizing power of our approach with a low cost approach such as random call pattern selection.

In order to accomplish the first goal, we used a network that has a well-known source of performance degradation. We tested our approach experimentally over a network spanning three geographically separated Avaya sites, namely Holmdel, Basking Ridge, and Denver. We refer to this network as the BHD network. This network contains an internet hop between New Jersey and Colorado that incurs significant delay. This hop constitutes a source of performance degradation in the BHD network. The ability to empirically to detect this link as the main source of performance problems constitute success.

In order to accomplish the second goal, we compared the results of our topology analysis based strategy, with the results generated by a call pattern generation strategy based on random selection. We refer to our call pattern generation strategy as topology analysis based call pattern generation strategy. Similarly, we refer to a call pattern generated by randomly selecting the endpoint pairs as random strategy based call pattern generation strategy. Random strategy based call patterns had comparable size to those of topology analysis based call patterns used in the empirical study and were exercised in an alternating fashion.

Using ExpertNet we first discovered the physical topology of the BHD network. Then we installed endpoint software on hosts throughout the network.

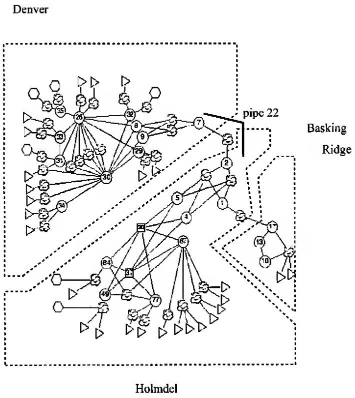


Figure 14. Physical Network Topology of the BHD network

Figure 14 shows the physical topology of the BHD network (layer-3 and layer-2 devices and their connections) with locations marked accordingly. The different shapes in the figure denote different types of network devices; specifically, circles, squares, triangles denote routers, switches and hosts running endpoint software, respectively. Clouds indicate areas of uncertainty. Specifically, there is a cloud between two devices which we know are connected but we are not sure about the details of the path connecting them. Hexagon denotes an undiscovered device whose IP address appears in a discovered router's route table. Note that each device may process packets for a number of vlans. Hence, each distinct device and vlan is considered as a distinct device. For simplicity, Figure 14 does not show all vlans at a physical device.

Given the topology and the endpoints, we reduced the topology as described in Section 3.1. Table 1 lists the 74 pipes computed for the BHD network topology and endpoint set. For simplicity, each pipe is assigned a number. The format of a pipe is *deviceid-vlanid* where *device* is a unique identifier for a device assigned by ExpertNet. The unique identifier for clouds has the form *cloud cloudid* where cloud id is a unique number assigned by ExpertNet to the clouds. Vlan id is the vlan number. In the absence of vlan tagging, vlan id has the value of "na". In Figure 14 an example pipe is shown. Note that this is the pipe with number 22 from Table 1. The cloud in this pipe 22 corresponds to the internet hop between New Jersey and

Colorado and constitutes the major source of performance degradation.

Pipe Id	Pipe Name
1	17-na . 13-140 . 10-140 . 10-na . cloud2-32-na
2	17-na . cloud2-31-na . 1-na
3	1-na . cloud2-15-na
4	4-na . cloud2-15-na
5	39-100 . 4-na
6	39-100 . 84-100 . 84-na
7	84-na . cloud2-19-na
8	37-200 . 4-na
9	37-200 . 87-200 . 87-na
10	87-na . cloud2-20-na
11	87-na . cloud2-16-na
12	39-100 . 49-100 . 49-na
13	49-na . cloud2-24-na
14	39-100 . 77-100 . 77-na
15	77-na . cloud2-22-na
16	87-na . cloud2-27-na
17	87-na . cloud2-25-na
18	77-na . cloud2-23-na
19	1-na . cloud2-41-na
20	2-na . cloud2-41-na
21	7-na . cloud2-42-na
22	2-na . cloud2-14-na . 7-na
23	30-8 . 9-na . cloud2-42-na
24	26-na . 26-8 . cloud2-43-na . 30-8
25	26-na . cloud2-1-na
26	26-na . cloud2-2-na
27	26-8 . cloud2-44-na . 30-8
28	26-8 . 29-na
29	29-na . cloud2-4-na
30	29-na . cloud2-5-na
31	30-8 . 30-na
32	30-na . cloud2-26-na
33	30-na . cloud2-6-na
34	34-na . 30-8 . 8-na . cloud2-42-na
35	34-na . cloud2-7-na
36	34-na . cloud2-8-na
37	26-8 . 32-na
38	32-na . cloud2-9-na
39	32-na . cloud2-36-na
40	26-8 . 35-na
41	35-na . cloud2-37-na
42	35-na . cloud2-38-na
43	26-8 . 33-na
44	33-na . cloud2-10-na
45	33-na . cloud2-11-na
46	30-8 . cloud2-12-na . 31-0 . 33-8 . 31-na
47	31-na . cloud2-39-na
48	31-na . cloud2-12-na
49	37-200 . 84-200 . 84-na
50	37-200 . 5-na
51	5-na . cloud2-15-na
52	17-na . cloud2-32-na
53	84-na . cloud2-49-na
54	87-na . cloud2-49-na
55	49-na . cloud2-49-na
56	77-na . cloud2-49-na
57	5-na . cloud2-41-na
58	37-200 . 49-200 . 49-na
59	37-200 . 77-200 . 77-na
60	26-7 . 26-na
61	26-7 . 8-na
62	8-na . cloud2-13-na
63	7-na . cloud2-13-na
64	2-na . cloud2-15-na
65	26-7 . cloud2-3-na . 30-7
66	29-na . 30-7
67	26-7 . 30-na
68	30-7 . 34-na
69	30-7 . 32-na
70	26-7 . 35-na
71	26-7 . 33-na
72	26-7 . 31-na
73	26-7 . cloud2-40-na . 30-7
74	30-7 . 30-na

Table 1. Pipe Index for the BHD Network

In accordance with our approach, we then selected a call pattern for the BHD network and the endpoint set using Algorithm Select_Matrix. The constraint that no endpoint can participate in more than 5 calls was imposed on the call pattern selection. Recall that we refer to this call pattern as the topology analysis based call pattern . We also employed random strategy to select call patterns to be exercised at every other time interval. Then ExpertNet injected synthetic calls by alternating between topology analysis based call patterns and random strategy based call patterns . This alternating fashion of data collection permits comparing the results obtained by the two distinct call pattern selection strategies. Various end-to-end statistics pertaining to these call patterns are stored in the ExpertNet database. In this section we focus only on end-to-end delay. The data collected corresponds to 5 days worth of data. The call durations and inter-call durations were the same for both strategies. Specifically, each call was 1 minute long and there was approximately a 10 minute interval between two consecutive executions of the same strategy.

The constrained call pattern selection on the BHD network selected a call pattern of size 55. Unfortunately, during the time intervals where topology analysis based call patterns were synthesized, some of the selected calls failed. Call failures also occurred during the time intervals where random strategy based call patterns were injected.

Figure 15 shows the distributions of the number of concurrent successful and failed calls for the time intervals pertaining to each call pattern selection strategy. In each plot, X axis shows the number of concurrent calls successful (or failed). Y axis shows the number of time intervals pertaining to a given number of concurrent successful (or failed) calls. Failure of a call between two endpoints prohibits data collection on the path between the endpoints and hence reduces network coverage. As a result, in this experiment out of 74 pipes, the topology based call patterns and the random strategy based call patterns did

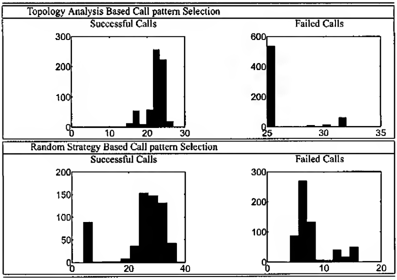


Figure 15. Summary of calls

not cover 25 and 23 pipes, respectively. 15 pipes appeared in both sets of uncovered pipes.

We generated two sets of link level estimates, each based on a different call pattern selection strategy. The link level estimates for the topology analysis based call pattern use only the data collected when the topology analysis based call pattern was exercised. Similarly, link level estimates for random strategy based call pattern use only the data collected when call patterns pertaining to this strategy were exercised. We solved the linear equations applicable at each time interval and used algorithm Compute_EstPaths to determine the estimable paths pertaining to the interval. In the next sections we present the results for each call pattern selection strategy.

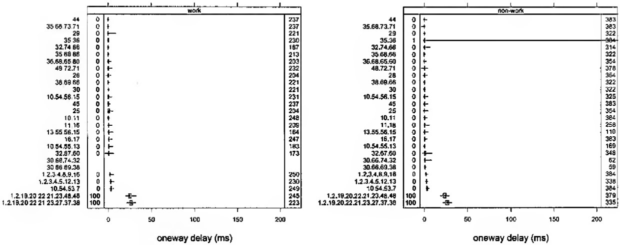
4.1 Results of Topology Analysis Based Call pattern Selection

Figure 16 is a plot of the delay distributions we computed for each estimable path that could be isolated in the network using the topology analysis based call pattern selection strategy. The time intervals pertaining to work hours (9am to 6 pm) are separated from non-work hours (6pm to 9am) in order to isolate varying network behavior during periods of different loads. The estimable paths, which are written in terms of pipe numbers, constitute the rows in Figure 16. For each path, a box plot summarizing the estimated delay distribution on that path over the data collection period is plotted. The box plot shows the 1^{st}, 25^{th}, 50^{th}, 75^{th}, and the 99^{th} percentile of the delay distribution. The dot is the 50^{th} percentile (the median), the box extends from the 25^{th} percentile (the first quartile) to the 75^{th} percentile quartile (the third quartile), and the whiskers extend to the 1^{st} and the 99^{th} percentiles. The rightmost column in each plot of the figure denotes the number of time intervals in which each path was estimable. The vertical line in in each plot of the figure corresponds to the oneway delay figure we consider as acceptable. In this case this figure is 10ms. The leftmost column in each plot indicates the percentage of time intervals in which the delay attributable to each path was unacceptable. Note that any path whose number of time intervals was 10 % less than the maximum time interval in which a path was estimable was ignored.

The oneway delay summaries of the paths in the figure indicate that except two paths, most of the paths in the figure have acceptable performance. We construct two sets of pipes C (*clean set*) and P (*problem set*) as follows: C contains pipes on the paths of Figure 16 work hours whose means are 0. P contains pipes on the paths of figure 16 work hours whose means are above the acceptable value of 10ms. Note that the same sets would be constructed using the plot for non-work hours. Pipes remaining in $P \setminus C$ denotes paths that need to be investigated further.

$C =$	{ 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 25, 26, 29, 30, 32, 35, 36, 38, 44, 45, 48, 53, 54, 55, 56, 60, 65, 66, 67, 68, 69, 71, 72, 73, 74 }
$P =$	{ 1, 2, 19, 20, 21, 22, 23, 27, 37, 38, 46, 48 }
$P \setminus C =$	{ 19, 20, 21, 22, 23, 27, 37, 46}

The results of topology analysis based call pattern selection identified the region formed by the 8 pipes in the set $P \setminus C$ as the source of performance degradation. The internet hop (pipe 22), which is the major cause of delay, is in the region.



05/22/2002 12:34 to 06/03/2002 11:53

05/22/2002 12:34 to 06/03/2002 11:53

Figure 16. Results with topology analysis based call selection (work and non-work hours)

Note that ideally, each row in the plots of Figure 16 pertains to a path composed of exactly one pipe. Clearly this is not the case in this figure. Recall that cycles in the graph and constraints imposed on call pattern selection prevent the selection of an optimum flow matrix. Furthermore, call failures have a cascading affect for topology analysis based call pattern selection. When no data is available of a path p, all pipes that p is used to estimate are no longer estimable.

4.2 Results of Random Strategy Based Call pattern Selection

Figure 17 and Figure 18 show the delay distributions for paths estimated using random strategy based call pattern selection for work and non-work hours, respectively. The template used for Figure 16 applies in this case. As before, the vertical line in the figure corresponds to the oneway delay figure of 10 ms which we consider as acceptable.

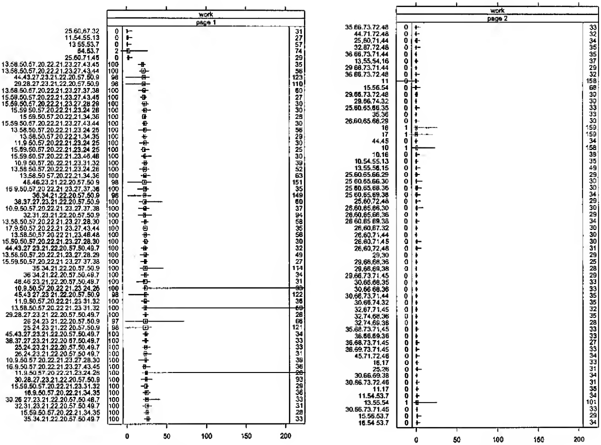
This strategy results with more estimable paths than those of topology analysis based selection. In Figure 17 and 18 many paths have delay distributions whose mean is different than 0. This is expected since any path involving pipe 22 will have non-zero delay.

As before, we construct two sets of pipes C and P using the work hours plot of Figure 17. The same sets would be obtained using Figure 18. Recall that, C is the set of pipes that are on paths of Figure 17 whose delay distributions are 0. P is the set of pipes that are on paths of Figure 17 whose delay distributions are not acceptable. Set $C \setminus P$ denotes the set of pipes that are possible causes for performance degradation. As shown below, random strategy implicated a region of 18 pipes as the source of performance degradation.

$C =$	{ 7 , 10 , 11 , 13 , 15 , 16 , 17 , 25 , 26 , 29 , 30 , 32 , 35 , 36 , 38 , 44 , 45 , 48 , 53 , 54 , 55 , 56 , 60 , 65 , 66 , 67 , 68 , 69 , 71 , 72 , 73 , 74 }
$P =$	{ 7 , 9 , 10 , 11 , 13 , 15 , 16 , 17 , 20 , 21 , 22 , 23 , 24 , 25 , 26 , 27 , 28 , 29 , 30 , 31 , 32 , 34 , 35 , 36 , 37 , 38 , 43 , 44 , 45 , 46 , 48 , 49 , 50 , 57 , 58 , 59 }
$P \setminus C =$	{ 9 , 20 , 21 , 22 , 23 , 24 , 27 , 28 , 31 , 34 , 37 , 43 , 46 , 49 , 50 , 57 , 58 , 59 }

4.3 Discussion

The results indicate that both strategies successfully identified the pipe incurring the largest delay in the network. However, both strategies had some false positives, that is the region they blamed for performance degradation included extra pipes .



05/22/2002 12:34 to 06/03/2002 11:53

05/22/2002 12:34 to 06/03/2002 11:53

Figure 17. Results with random call selection (work hours)

Table 2 below summarizes the results: In the table, first column shows the number of pipes each strategy covered throughout the data collection period. The second column shows network coverage computed as the ratio of the number of pipes covered with respect to the total number of pipes , which is 74. The third column shows the size of the problem region each strategy predicted as the cause of performance degradation. The unit of region size is the number of pipes . The last column is a measure of how accurately each strategy pinpointed the source of degradation showing how the blamed region size compares to the actual problem region size. This measure is computed for each strategy as $1 - \frac{blamed\_size - actual\_size}{total\_size}$. Recall that the actual problem region size is 1 pipe and the total size is 74 pipes . Since both strategies have only false positives, this simple measure of accuracy is sufficient in this case. Note that accuracy reduces as the number of false positives increase.

Network Coverage

Table 2 indicates that random strategy based call patterns have a slightly better network coverage. This is expected since topology based call patterns exercise the same static pattern with no redundancy at every time interval. This makes them more vulnerable to failed calls. Random strategy based call patterns have more redundancy over a long period of time. However, the empirical results indicate that in this experiment, over a long period of time (5 days) there was not much of a difference in the network coverage between the two strategies.

Accuracy

In terms of narrowing down the region to be blamed for performance degradation, topology based approach resulted with a much smaller region as shown in Table 2. In order to understand the nature of the performance problem, it is necessary

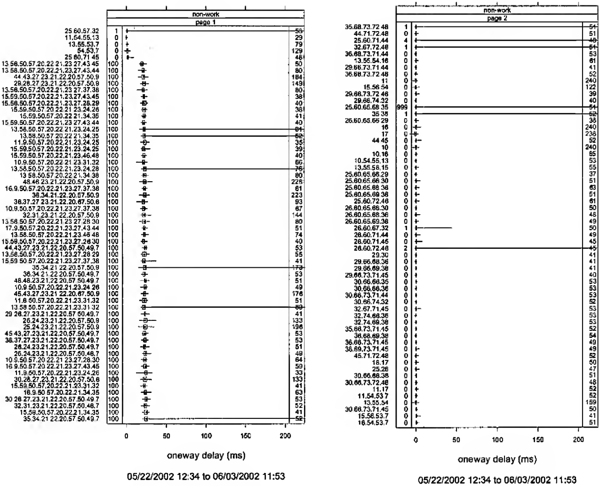


Figure 18. Results with random call selection (non-work hours)

to analyze the blamed region in detail. Hence pinpointing this region as accurately as possible saves significant amount of work. Topology analysis based call patterns provide 91% accuracy wrt 77% provided by the random strategy. Note that if the call failures did not occur during data collection, the accuracy of topology analysis based call patterns would have been almost 100%. In addition, the effort to compute the region is significantly smaller for topology analysis based as shown in the earlier section.

5 Conclusion

In this paper we describe an approach to understanding link level QoS behavior in a network. We apply our approach to locate the sources of performance problems from the perspective of QoS sensitive applications such as IP Telephony. Our approach is based on collecting end-to-end measurements from the network under consideration with the use of our prototype tool ExpertNet™and using these measurements to predict link level estimates. ExpertNet™is a tool that collects various measurements from a network including topology information, periodic traffic measurements from network devices via SNMP, and end-to-end performance measurements with the use of software agents installed in the network. Our approach starts with analyzing the network topology with respect to the locations of software agents for collecting measurements. We determine a set of network paths that can be assigned performance estimates such that this set is composed of as short and as disjoint set of paths as possible. Our measurement collection strategy determines where in the network to collect end-to-end

Strategy	# pipes covered	Network Coverage	Blamed Region Size (Actual problem region size = 1)	Accuracy
Topology Analysis Based	49	66%	8	91%
Random Strategy Based	51	69%	18	77%

Table 2. Summary of Results

measurements such that we can determine performance estimates for the paths in the selected set. Using ExpertNet^{TM}, we collect such measurements. Finally, we analyze the collected data in light of the topology and infer performance estimates on links or groups of links.

We present an algorithm that analyzes the network topology and provides a set of paths to which performance estimates can be assigned. This algorithm also provides the set of end-to-end paths on which measurements should be collected to infer link level performance estimates for paths in the selected set. We formally prove that when the network topology is a tree, this algorithm determines the *optimum* set of paths. Note that our approach considers the physical topology hence it is expected that the topology is either a tree or close to being a tree.

We present the results of an empirical study conducted on a real network spanning three geographically separated Avaya sites. The main goal of our study is to determine the accuracy of our approach in narrowing down the problem region to be blamed for performance degradation. Informally, accuracy is a measure of the ability to pinpoint the source of performance degradation. The empirical results indicate that our approach efficiently determines the links in the network incurring the most significant delay. Our approach provides 91% accuracy compared with a fully random strategy which provides 77% accuracy.

References

[1] M. Bearden, L. Denby, B. Karacali, J. Meloche, and D. T. Stott. "Assessing Network Readiness for IP Telephony,". To Appear in the Proc. of the 2002 IEEE Intl. Conf. on Communications (ICC-02), 2002.

EXHIBIT 2

From: Bengi Karacali [karacali@avaya.com]
Sent: Friday, September 20, 2002 10:37 AM
To: Martina Sharp
Subject: ALR-2002-037 files

ALR-2002-037-Cover.doc alr-2002-037.pdf

Hi Tina,

Attached are the cover sheet and the paper in pdf format for ALR-2002-037.

Thanks.
Bengi

EXHIBIT 3

Teresa Hamlin

From: "Chesal, Lourdes (Lourdes)" <lchesal@avaya.com>
To: <nyoffice@rml-law.com>
Sent: Wednesday, April 23, 2003 4:08 PM
Attach: patent_summary.doc
Subject: FW: New patent possibilities for ExpertNet technology

Teresa,

Per our telephone conversation a few minutes ago, here is the submission material. I will fax the inventor details this afternoon. I have given it Submission Number 503021.

Thanks,

Lourdes Chesal
Docket Administrator
Avaya Inc.
307 Middletown-Lincroft Rd.
Lincroft, NJ 07738
lchesal@avaya.com <mailto:lchesal@avaya.com>

-----Original Message-----
From: Dinicola, Brian K (Brian)
Sent: Wednesday, April 23, 2003 11:43 AM
To: Chesal, Lourdes (Lourdes)
Subject: FW: New patent possibilities for ExpertNet technology

Lourdes, can you assign an IDS number to this patent proposal. The title would be "Enhanced System for Isolating Sources of Network Performance Problems."

Inventors are: Lorraine Denby and Bengi Karacali

Send to Joe Ryan of Ryan and Mason

Brian

-----Original Message-----
From: Karacali, Bengi (Bengi)
Sent: Wednesday, April 23, 2003 10:43 AM
To: Dinicola, Brian K (Brian); Denby, Lorraine (Lorraine); Landwehr, James M (Jim); Kintala, Chandra M (Chandra)
Cc: Karacali, Bengi (Bengi)
Subject: New patent possibilities for ExpertNet technology

Hi Brian,

As per our discussion yesterday regarding patent possibilities for an advanced feature of ExpertNet, attached is a brief summary describing this feature at a high level.

Thanks,
-Bengi

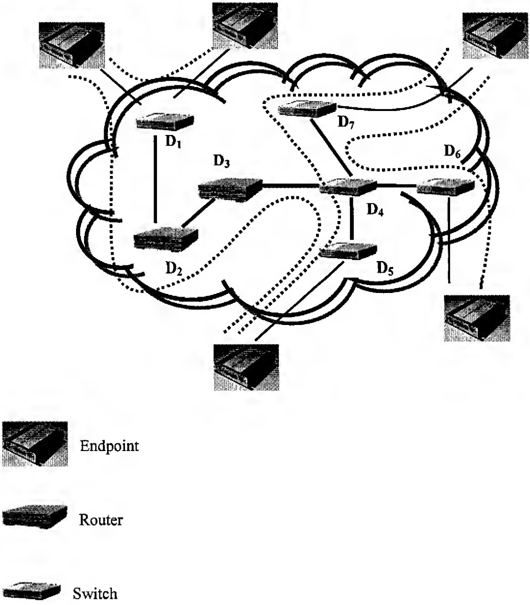
ExpertNet^{SM} is a tool developed at Avaya for assessing network readiness for IP Telephony. Avaya Network Consulting Services currently employs this tool to assess customer networks prior to deployment of IP Telephony for the purpose of identifying performance problems and their sources. It encompasses various novel patent-pending ideas. Advanced features are integrated to new releases of ExpertNet^{SM} to distinguish it further from competitors. Among the new features is the ability to automatically determine the location of performance problems in a network.

This feature relies on the data collected by ExpertNet^{SM} to automatically isolate the sources of performance problems in the network. For instance, consider the network shown in Figure 1. This network consists of seven network devices as marked on the figure. During assessment, ExpertNet^{SM} places endpoints in this network as indicated by the icons in the figure and collects measurement between these endpoints by injecting IP Telephony traffic. Dotted lines in the figure indicate the paths traversed by the injected traffic. ExpertNet^{SM} measures the quality of service (QoS) achieved by the injected traffic in terms of metrics such as delay and loss and determines the network devices and links traversed by the injected traffic. For example, a performance problem between devices 3 and 4 indicated with a red link in Figure 1, causes performance problems for all synthetic traffic traversing it. ExpertNet^{SM} observes this problem on the end-to-end path level, meaning all injected traffic traversing this link shows high delay or loss figures. In order to ascertain that the problem is actually between devices D_3 and D_4, an engineer would manually study the significant amounts of data collected by ExpertNet^{SM}.

The new feature simplifies this analysis by determining individual quality figures for the network links. This feature automatically estimates the delay or loss on each link using the integrated data collected by ExpertNet^{SM}, specifically end-to-end quality data and the network topology. In the network shown in Figure 1, this feature would immediately identify that the link between devices 3 and 4 is the source of performance degradation. Having located the source of the problem, the engineer would then identify the cause of the problem.

Another advantage of this feature is the ability to guide traffic injection such that the data collection and the analysis can be conducted in a scalable manner. For example, in the network shown in Figure 1, it is possible to inject traffic between every pair of 5 endpoints to determine that the problem is between devices D_3 and D_4. However, it is possible to reach the same conclusion using much less number of endpoint pairs in traffic injection.

Figure 1


D₁
D₃
D₇
D₆
D₄
D₂
D₅
Endpoint
Router
Switch